Gliwice 18.02.2019

Karol Dudek
Semester 3
Group 3

Computer Programming 3
Banking System

Author: Karol Dudek
Tutor: mgr in. Anna Gorawska

Introduction:

This program is a simulator of the banks database. It contains informations about Different type of accounts in bank and about Clients. The data is being stored in textfiles, so program doesn't lose data. In program the data is stored and processed in vectors in class database.

Project Analysis:

To create program first I had to come up with idea of the objective structure I want to base it on. By it I mean I had to decide what classes will I create, how will they communicate with each other and what are gonna be their responsibilities.

This way I designed it in following way:

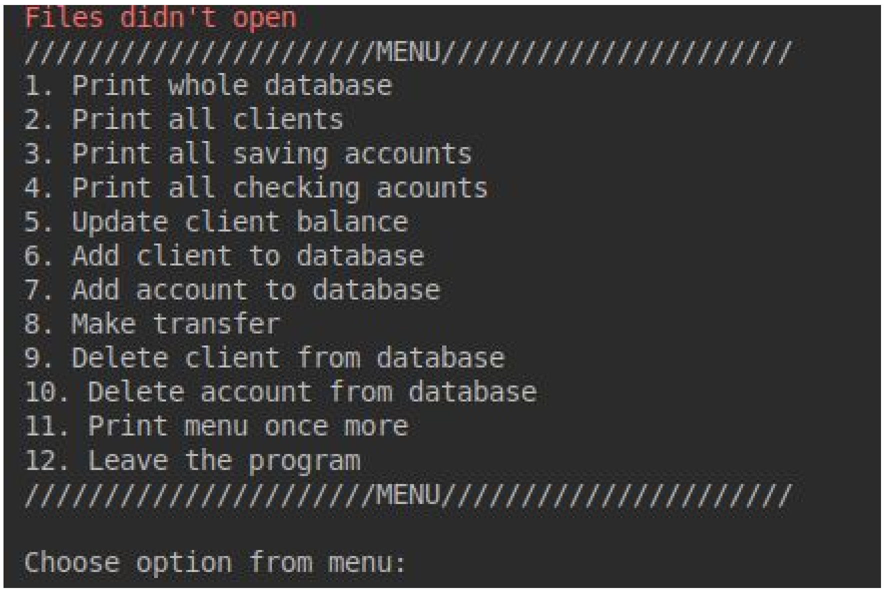
- Person - class representing person in database. This person, of course, is a client in bank and is able to create accounts in it.
- BankAccount - class representing bank account. From it two classes representing different bank accounts are deriving. - CheckingAccount and SavingsAccount.
- Database - the most important class responsible for whole system of database.

I also designed few global functions like getPositiveNumber() that makes sure that user provides positive number, it is useful in many places in program, printMenu() printing menu with options and handleOptions() running these options.

Also to prevent situation when program tries to read from files that do not exist I included exception handling that takes care of it. In case of exception being thrown while creating database the blank files are created and then new Database object is being created without starting data.

External Specification:

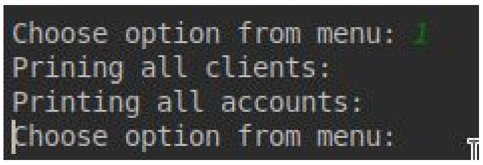
On the first run of the program following console window appears:



```
Files didn't open
//////////////////////MENU//////////////////////
1. Print whole database
2. Print all clients
3. Print all saving accounts
4. Print all checking acounts
5. Update client balance
6. Add client to database
7. Add account to database
8. Make transfer
9. Delete client from database
10. Delete account from database
11. Print menu once more
12. Leave the program
//////////////////////MENU//////////////////////

Choose option from menu:
```

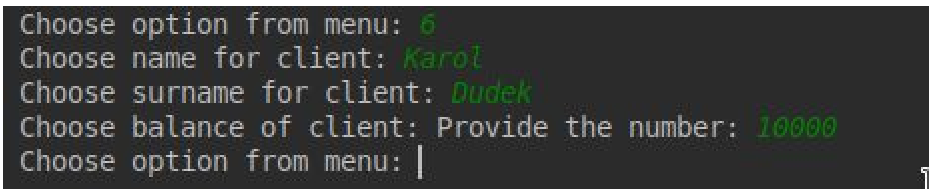
If the files would have existed the red information wouldn't be printed. Now program created empty files for further use.

Then we can choose different options, for example, choosing first option will result in following output because there is not data yet in database.



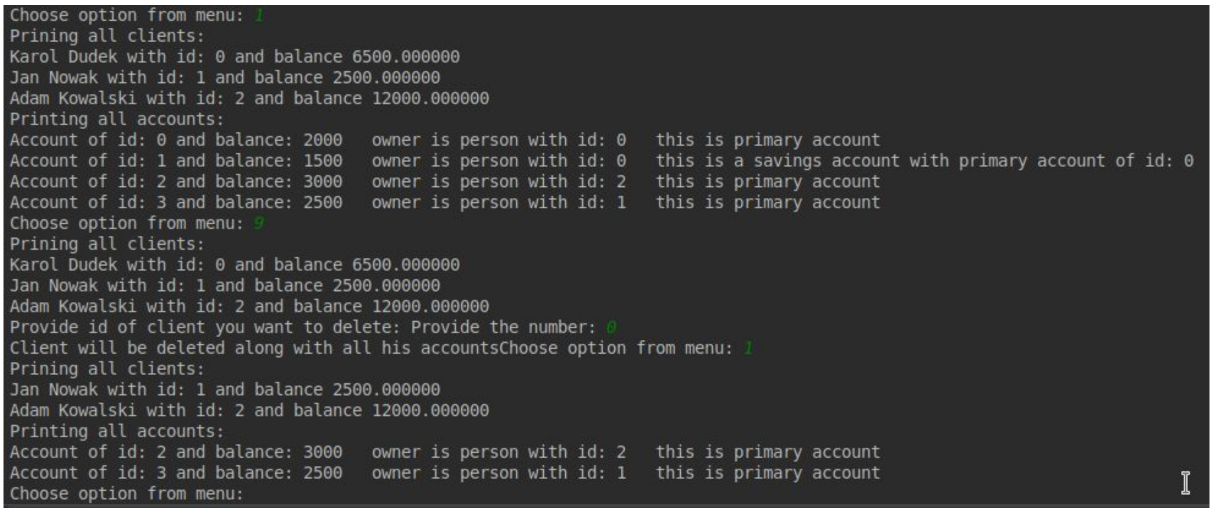
```
Choose option from menu: 1
Prining all clients:
Printing all accounts:
Choose option from menu:
```

Options 6 and 7 are handled by simple creator of object that looks this way.



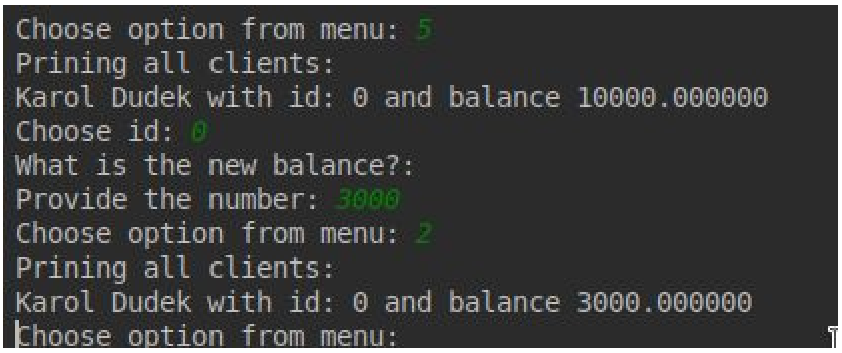
```
Choose option from menu: 6
Choose name for client: Karol
Choose surname for client: Dudek
Choose balance of client: Provide the number: 10000
Choose option from menu: |
```

To delete any account or client (options 9 and 10) we also have to fill in short 'form'. However deleting client will results in deleting all his accounts. Also deleting CheckingAccount will result in deleting savings account corresponding to it. Of course all money is first being transferred to its owner balance.



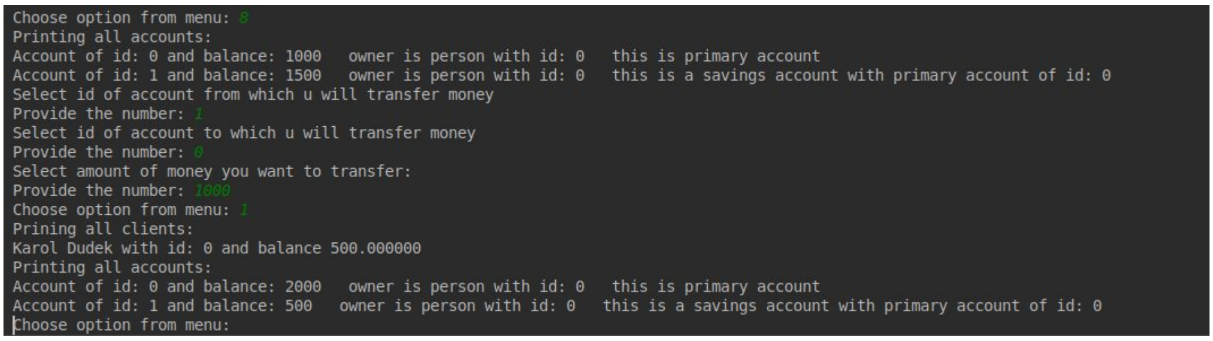
```
Choose option from menu: 1
Prining all clients:
Karol Dudek with id: 0 and balance 6500.000000
Jan Nowak with id: 1 and balance 2500.000000
Adam Kowalski with id: 2 and balance 12000.000000
Printing all accounts:
Account of id: 0 and balance: 2000   owner is person with id: 0   this is primary account
Account of id: 1 and balance: 1500   owner is person with id: 0   this is a savings account with primary account of id: 0
Account of id: 2 and balance: 3000   owner is person with id: 2   this is primary account
Account of id: 3 and balance: 2500   owner is person with id: 1   this is primary account
Choose option from menu: 9
Prining all clients:
Karol Dudek with id: 0 and balance 6500.000000
Jan Nowak with id: 1 and balance 2500.000000
Adam Kowalski with id: 2 and balance 12000.000000
Provide id of client you want to delete: Provide the number: 0
Client will be deleted along with all his accountsChoose option from menu: 1
Prining all clients:
Jan Nowak with id: 1 and balance 2500.000000
Adam Kowalski with id: 2 and balance 12000.000000
Printing all accounts:
Account of id: 2 and balance: 3000   owner is person with id: 2   this is primary account
Account of id: 3 and balance: 2500   owner is person with id: 1   this is primary account
Choose option from menu:
```

On demand we can also use option 11 to print menu once more or choose option 12 to close the program.
There is also option 5 that updates the balance of the client.



```
Choose option from menu: 5
Prining all clients:
Karol Dudek with id: 0 and balance 10000.000000
Choose id: 0
What is the new balance?:
Provide the number: 3000
Choose option from menu: 2
Prining all clients:
Karol Dudek with id: 0 and balance 3000.000000
Choose option from menu:
```

And option 8 for transfering money



```
Choose option from menu: 8
Printing all accounts:
Account of id: 0 and balance: 1000   owner is person with id: 0   this is primary account
Account of id: 1 and balance: 1500   owner is person with id: 0   this is a savings account with primary account of id: 0
Select id of account from which u will transfer money
Provide the number: 1
Select id of account to which u will transfer money
Provide the number: 0
Select amount of money you want to transfer:
Provide the number: 1000
Choose option from menu: 1
Prining all clients:
Karol Dudek with id: 0 and balance 500.000000
Printing all accounts:
Account of id: 0 and balance: 2000   owner is person with id: 0   this is primary account
Account of id: 1 and balance: 500   owner is person with id: 0   this is a savings account with primary account of id: 0
Choose option from menu:
```

Internal Specification:

This program has a lot of functions and methods. However many of them explain what they do pretty much on their own by their name like for example addClient() - adding client. Because of that I will put here only these methods that are more complex or especially important.

For class Database:

void addClient(), void addAccount()

these methods do pretty much as their names say. They provide some forms for user to create client or account in bank. After creation of object they adress of it is pushed to corresponding vector/vectors of pointers.

void readFromFiles(), void saveToFiles()

these methods are very imortant because they are responsible to actually save our database to the hard drive. When database is being created constructor calls first method. Destructor calls second one.

For class Person

Actually the only methods worth to be mentioned here are constructors. During object creation process the ID has to be assigned to every each of them. First constructor assigns in automatically, second one used in reading from files allows user to choose the id.
In case of 2nd one important thing was to make sure that nextID variable will be updated correctly in case of running program few times and deleting entries of early ids.

For class BankAccount and classes deriving from it:

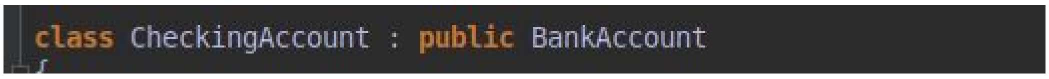
there is important variable of type **type**. It is enum defined by me to distinguish type of the account. It is used many times in distinguishing the type of account if it is not exactly known by its type because for example it is stored in vector of BankAccount objects.

Then there is a method virtual void topUp() that is overriden in deriving classes.It tops up account by amount of money but in the way specific to type of account. Along with it there is self-explainatory virtual withdraw(), and also virtual toString() and printAccountInfo() that prints (or returns) string representing information about account.

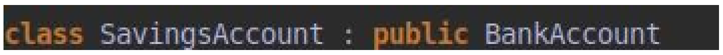
Where topics are used:

Inheritance:

Inheritance is used in program by creating classes SavingsAccount and CheckingAccount. Both of them share some characteristics, for example both have their balance and owner, but they behave in different way. That's why I decided to use inheritance here instead of creating 2 completely separate classes looking halfway the same.



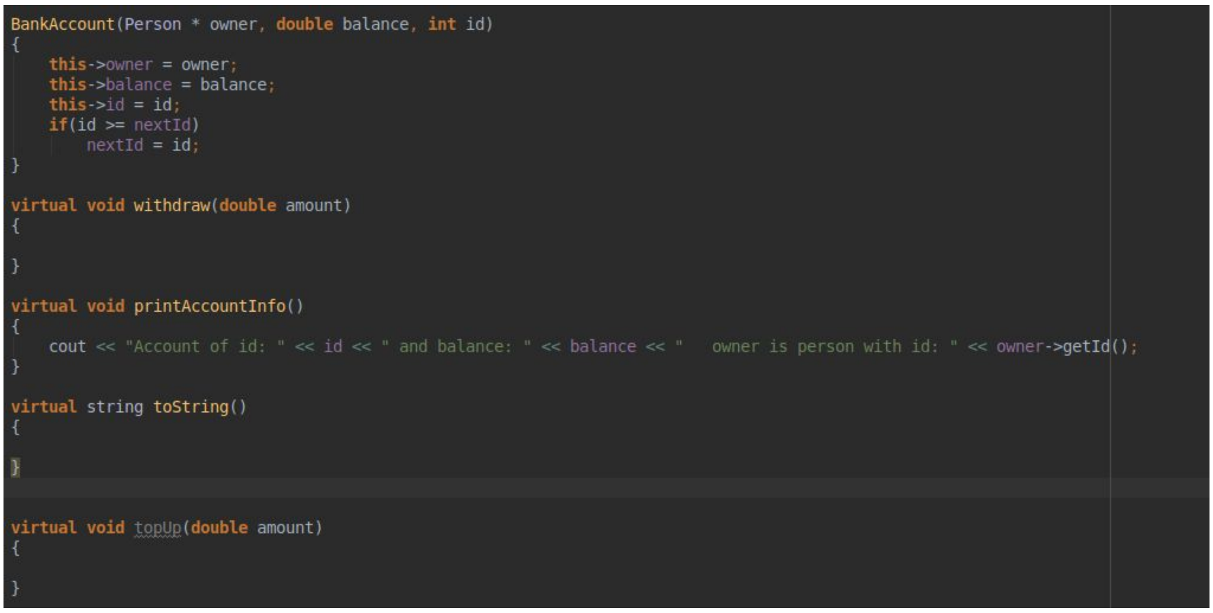
```
class CheckingAccount : public BankAccount
```



```
class SavingsAccount : public BankAccount
```

Polymorphism

It is used for creating 4 virtual methods inside BankAccount class. These are methods shown on the screenshot. Sometimes, like in case of printing method the difference is subtle and it uses base class implementation only adding some short string. Sometimes like in case of withdraw the difference is bigger as the way in which money is transferred - especially source of it - is different.



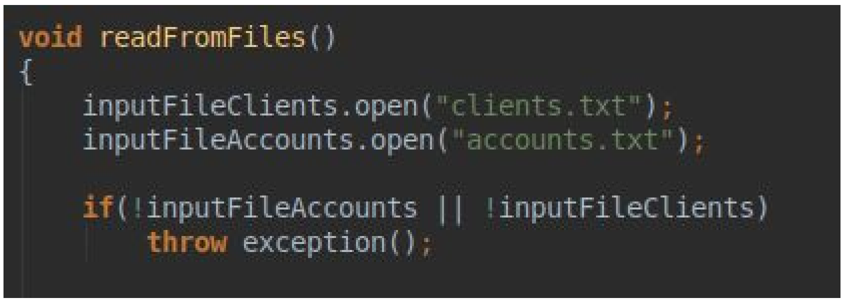
```
BankAccount(Person * owner, double balance, int id)
{
    this->owner = owner;
    this->balance = balance;
    this->id = id;
    if(id >= nextId)
        nextId = id;
}

virtual void withdraw(double amount)
{

}

virtual void printAccountInfo()
{
    cout << "Account of id: " << id << " and balance: " << balance << "   owner is person with id: " << owner->getId();
}

virtual string toString()
{

}

virtual void topUp(double amount)
{

}
```

IOStreams:

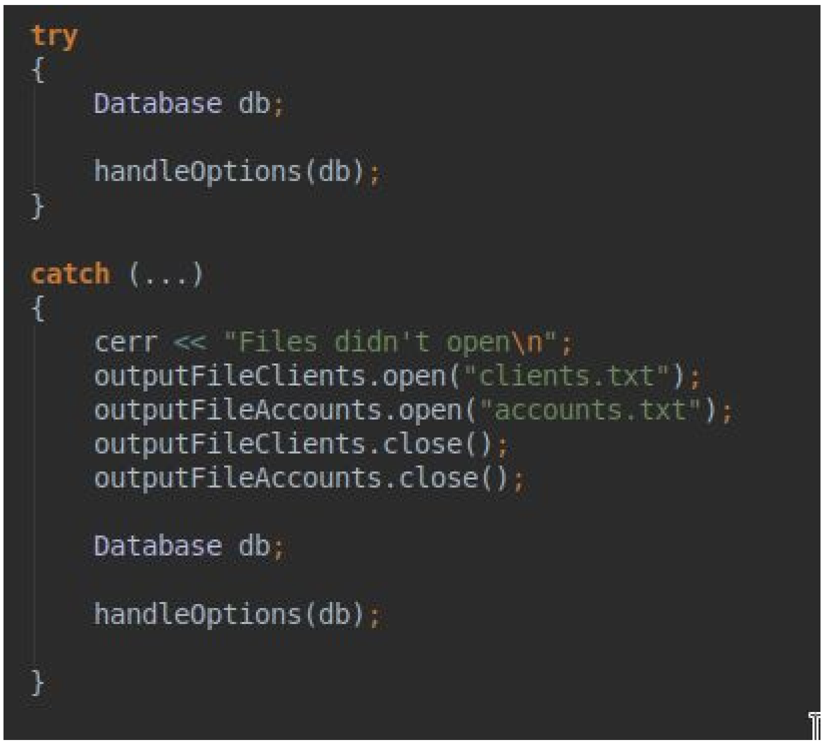
This probably doesn't need explanation. In whole program information is printed to console and inputted by user. Also, the information is saved and read from a file by readFromFiles and saveToFiles methods in Database class. That's also the place for:

Exceptions:

These are used in readFromFiles method. Exception is thrown if files with data couldn't be opened so there is no problem with trying to do some things with files that do no exist. Instead the files are created and then all processes come as they should.



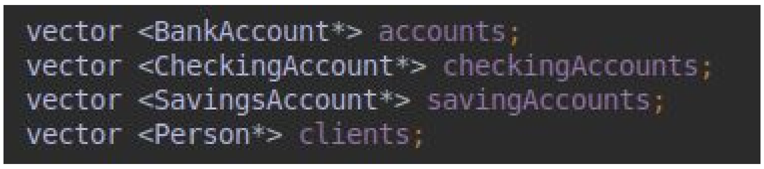
```
void readFromFiles()
{
    inputFileClients.open("clients.txt");
    inputFileAccounts.open("accounts.txt");

    if(!inputFileAccounts || !inputFileClients)
        throw exception();
```



```
try
{
    Database db;

    handleOptions(db);
}

catch (...)
{
    cerr << "Files didn't open\n";
    outputFileClients.open("clients.txt");
    outputFileAccounts.open("accounts.txt");
    outputFileClients.close();
    outputFileAccounts.close();

    Database db;

    handleOptions(db);

}
```

Also if we count it as using this topic, my program uses TEMPLATES:

It uses it by using vectors from STL - standard template library. These vectors store all data from database and are one of the most important elements of the class.

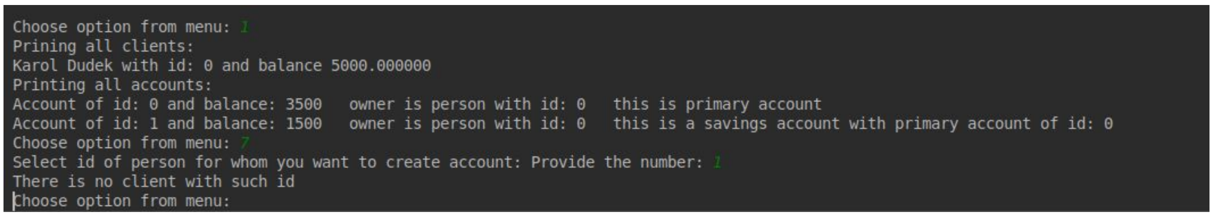


```
vector <BankAccount*> accounts;
vector <CheckingAccount*> checkingAccounts;
vector <SavingsAccount*> savingAccounts;
vector <Person*> clients;
```

Testing:

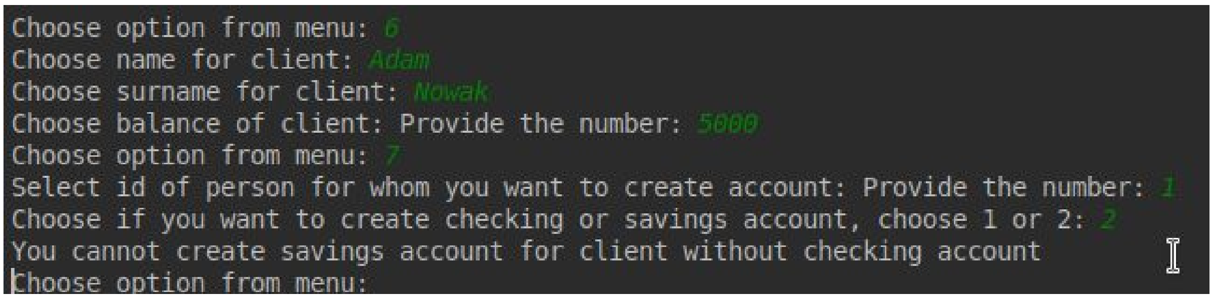
Most of the testing I've shown in external specification section. So here I will show only the moments that are responsible for handling errors and logic.

We can't create account for non-existing client. Similar cases are handled the same way.



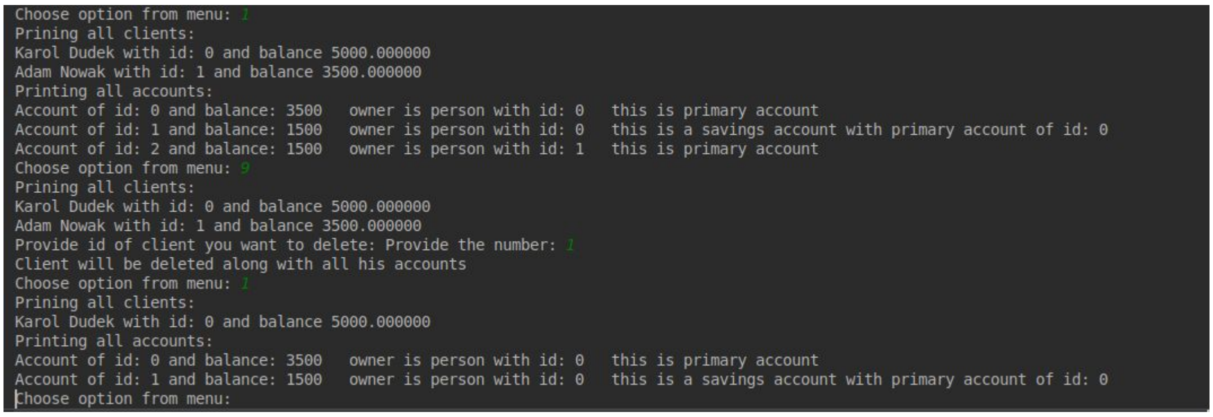
```
Choose option from menu: 1
Prining all clients:
Karol Dudek with id: 0 and balance 5000.000000
Printing all accounts:
Account of id: 0 and balance: 3500   owner is person with id: 0   this is primary account
Account of id: 1 and balance: 1500   owner is person with id: 0   this is a savings account with primary account of id: 0
Choose option from menu: 7
Select id of person for whom you want to create account: Provide the number: 1
There is no client with such id
Choose option from menu:
```

We can't create savings account for client without checking account



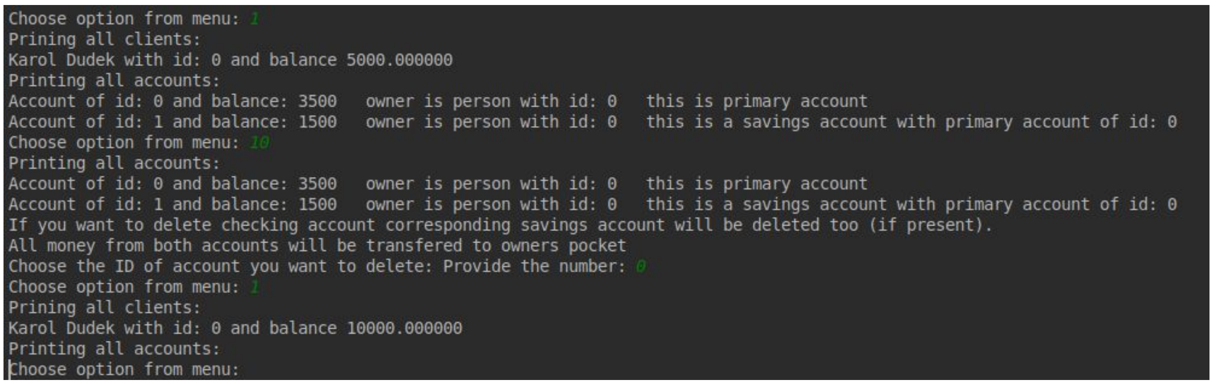
```
Choose option from menu: 6
Choose name for client: Adam
Choose surname for client: Nowak
Choose balance of client: Provide the number: 5000
Choose option from menu: 7
Select id of person for whom you want to create account: Provide the number: 1
Choose if you want to create checking or savings account, choose 1 or 2: 2
You cannot create savings account for client without checking account
Choose option from menu:
```

If we decide to delete client all his accounts are deleted with him.



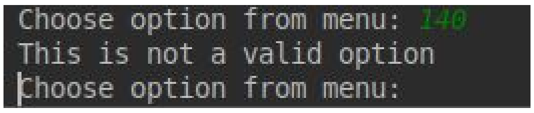
```
Choose option from menu: 1
Prining all clients:
Karol Dudek with id: 0 and balance 5000.000000
Adam Nowak with id: 1 and balance 3500.000000
Printing all accounts:
Account of id: 0 and balance: 3500   owner is person with id: 0   this is primary account
Account of id: 1 and balance: 1500   owner is person with id: 0   this is a savings account with primary account of id: 0
Account of id: 2 and balance: 1500   owner is person with id: 1   this is primary account
Choose option from menu: 9
Prining all clients:
Karol Dudek with id: 0 and balance 5000.000000
Adam Nowak with id: 1 and balance 3500.000000
Provide id of client you want to delete: Provide the number: 1
Client will be deleted along with all his accounts
Choose option from menu: 1
Prining all clients:
Karol Dudek with id: 0 and balance 5000.000000
Printing all accounts:
Account of id: 0 and balance: 3500   owner is person with id: 0   this is primary account
Account of id: 1 and balance: 1500   owner is person with id: 0   this is a savings account with primary account of id: 0
Choose option from menu:
```

If we delete checking account the saving account is deleted with it and all money is transfered to client.



```
Choose option from menu: 1
Prining all clients:
Karol Dudek with id: 0 and balance 5000.000000
Printing all accounts:
Account of id: 0 and balance: 3500   owner is person with id: 0   this is primary account
Account of id: 1 and balance: 1500   owner is person with id: 0   this is a savings account with primary account of id: 0
Choose option from menu: 10
Printing all accounts:
Account of id: 0 and balance: 3500   owner is person with id: 0   this is primary account
Account of id: 1 and balance: 1500   owner is person with id: 0   this is a savings account with primary account of id: 0
If you want to delete checking account corresponding savings account will be deleted too (if present).
All money from both accounts will be transfered to owners pocket
Choose the ID of account you want to delete: Provide the number: 0
Choose option from menu: 1
Prining all clients:
Karol Dudek with id: 0 and balance 10000.000000
Printing all accounts:
Choose option from menu:
```

Also, if we choose the option that is not present in menu we will be also notified and asked to choose again:



```
Choose option from menu: 140
This is not a valid option
Choose option from menu:
```

Conclusions:

Creating this program made me learn and understand completely new concept of designing a program. I’m talking of course about OOP. I think that database-like project was a great way to introduce it without too complicated confusing things. This way I learned a lot and for sure I will continue to use OOP in my further projects.,